Vol 8. No. 1 – March, 2015
African Journal of Computing & ICT

www.ajocict.net

A User-Friendly Query Interface Based on Best Position Algorithm for Distributed Databases

C. Ugwu & M. Abuh
Department of Computer Science
University Of Port-Harcourt
Port-Harcout , Nigeria.
chidiebereugwu@yahoo.com, abuhnature@yahoo.com;

ABSTRACT

The need for a user friendly query interface that helps non expert database users that use database effectively in a distributed environment was identified as a gap to be filled. However, existing systems are still unable to provide a user friendly platform for querying databases without the need to type codes, and an obvious implication of these, is the reduced exploitation of the ability of database systems especially in a distributed environment, by non technical users. This paper adopts an object oriented methodology and presents an approach that overcomes some of these flaws by using the Best Position Algorithm (BPA), which is an efficient algorithm for the problem of answering top-k queries over sorted lists in a distributed database. Implementation of the system was done using Java Programming language and Microsoft SQL Server 2005 as the back-end and driven by JDBC API. The results were commonly needed queries fetched from a database and presented to the user just at the click of a mouse. Furthermore, the automatically generated query statement executed is presented to the user so as to educate and help build competence. These implies that users can now run queries and perform tasks on a database even in a case of complete ignorance of the underlying query technicalities with few or no manually written query, rather just at the click of a mouse button.

Keywords: User-Friendly Query Interface (UQI), Natural Language Interface to Databases, Best Position Algorithm (BPA).

African Journal of Computing & ICT Reference Format:
C. Ugwu & M. Abuh (2015): A User-Friendly Query Interface Based on Best Position Algorithm for Distributed Databases.
Afr J. of Comp & ICTs. Vol 8, No. 1. Pp 219-224.

1. INTRODUCTION

Natural Language Query Interface to Database (NLIDB) systems has made it easy to manipulate database systems without the need for the user to use formal query languages (Gabriel et al, 2013, Davis, 2014), such as SQL. Database query languages can be difficult to non-expert users and learning these formal queries takes a lot of time. Query interfaces are meant to support users in formulating a precise query against a database described by a specific data model. Queries are specified by means of special purpose query languages, where a query language is a set of formally defined operators allowing requests to be expressed to a database. (Kacprzyk, and Zadrozny, 2001). By executing a query, the user expects that the produced results extracted from the stored data are coherent with the intended meaning of the request. The most widely used database query languages have been programming languages which require knowledge about language syntax, technical background, and information of both the system application domain and its interaction mechanisms. Such languages do not help to understand the meaning of data, nor do they provide any guidance in satisfying the user's needs. In general, they do not fulfill the requirements of user friendliness and ease of use (Oussama, 2001, Hallet, 2006).

But lately, there is an overwhelming need for non-expert users to query relational databases in their natural language using linguistic variables and terms instead of working with the values of the attributes. As a result, intelligent databases and interfaces have emerged, which provides expanded and more flexible options for manipulating queries.

2. RELATED LITERATURE

Our work has been inspired by a number of works available in the literature related to intelligent aspects of database systems. The field of intelligent database and information systems has achieved remarkable growth in the last few decades. Researches in the area of intelligent query processing interface in a distributed database, has started to increase the efficiency of retrieving and exchanging information between database applications and users, and thus have made the exploration of databases much more embraced. Benharzallah, et al., (2001), proposed an efficient query processing approach for semantic interoperable information systems, they also proposed a generic multi agent architecture that supports the approach.

The approach consists in the exploitation of intelligent agents for query reformulation and the use of a new technology for the semantic representation. The algorithm is self-adapted to the changes of the environment, offers a wide aptitude and solves the various data conflicts in a dynamic way; it also reformulates the query using the schema mediation method for the discovered systems and the context mediation for the other systems. Neelu, et al. (2009), proposed an intelligent layer for database which is responsible for manipulating flexible queries. Initially, the flexible queries from users in their natural language are submitted to intelligent layer and this layer converts the amorphous query into a structured SQL query. The shaped query is executed and the results are presented to the user. Afterwards, on the basis of results, feedback and the acceptance or rejection of the results are requested from the user. It enables the design of a knowledge based self learning system based the values obtained from user, which will aid the selection of appropriate SQL query, when a same flexible query is issued in the future. The experimental results demonstrate the effectiveness of the proposed intelligent database system.

Khayut, et al.(2014), proposed the data, information and knowledge based technology of Smart/Intelligent User Interface (IUI) design, which interacts with users and systems in natural and other languages, utilizing the principles of Situational Control and Fuzzy Logic theories, Artificial Intelligence, Linguistics, Knowledge Base technologies and others. The proposed technology of IUI design was defined by multi-agents of (a) Situational Control of data, information and knowledge, (b) modeling of Fuzzy Logic Inference, (c) Generalization, Representation and Explanation of knowledge, (d) Planning and Decision-making, (e) Dialog Control, (f) Reasoning and Systems Thinking, (g) Fuzzy Control of organizational unit in real-time, fuzzy conditions, heterogeneous domains, and (g) multi-lingual communication under uncertainty and in Fuzzy Environment.

In (Ndueso et al, 2014) An Intelligent layer for Database was designed which is responsible for manipulating flexible queries. Initially, the flexible queries from users in their natural language are submitted to intelligent layer and this layer converts the amorphous query into a structured SQL query. The shaped query is executed and the results are presented to the user. Afterwards, on the basis of results, feedback and the acceptance or rejection of the results are requested from the user. It enables the design of a knowledge based self learning system based the values obtained from user, which will aid the selection of appropriate SQL query, when the same flexible query is issued in the future. However, this requires users to formulate natural language queries in an organized manner, so as to enable the intelligent layer recognize, read the query, and parse. Another problem is the time taken to match the natural language queries to appropriate SQL commands, as well as the time taken to formulate such queries(Ben et al, 2014 and Ben et al, 2013).

Another problem is the requirement of using additional knowledge to extract meaningful information, the input can have many choices and it is not easy to choose the correct choice among target representations (one-to-many mappings), the complexity of mapping in NL sentences if you change a single word, the entire structure can be changed, which is called the quantifier scoping problem. Words such as “the,” “each,” or “what” can have several meanings in different situations production rules for the possibly introduced queries. Also there is another of the identification of tables required to build. (Nittaya K. and Kittisak K., 2012).

All these works have been major breakthroughs, but this has also created another complex problem as to how the natural language queries should be formulated so as to be parsed by the compiler as well as the right structure of such queries. This work eliminates such flaws by not allowing the user go through the task of thinking of how to ask questions or queries, but automatically generates the queries for the users to select just by the click of the mouse. In this work, we solved some of these problems by designing a framework where users can access different databases in a server and run queries just by clicking the mouse. JDBC API assists in fetching all the tables and columns in a particular database server and displays the results to the user for selection. We are focused at building a bridge between database query technologies and non expert users in a distributed environment. The idea is to have a query interface that enables the users to access heterogeneous data sources by means of an intelligent agent (JDBC API) without having to write many queries using the best position algorithm. The query interface supports the users in the task of formulating precise and accurate query without an idea of the complexities and technicalities required to manually write such codes

3. MATERIALS AND METHODS

In this paper, we adopted the object oriented model in developing the User Friendly Interface. The programming languages used was MS SQL Server 2005 and Java because of their robustness interoperability. The database connectivity and manipulations were driven by an intelligent tool called JDBC API. JDBC is a Java-based data access technology (Java Standard Edition platform) from Oracle Corporation, and we followed the best position algorithm (BPA) for top-K queries*)* (Akbarinia, 2007). In the existing system architectures, the system will check if user question has SQL built in functions by using semantic dictionary data if there exists, it will get corresponding function name to determine which function category it belongs to in the system, it covers some categories such as mathematical and statistics functions (Count, Sum, Max, Min, Avg, Mean, etc). It searches for some words in user question and they are mapped into the semantic dictionary with corresponding word in lexical dictionary for example how many in semantic dictionary will be converted into count in lexical dictionary, as it already has predefined data in semantic and lexical dictionaries.

Though this system enables the querying process to be more user friendly, and eases the whole querying process, it does not completely eliminate the issues experienced by users in querying, rather it creates its own problem of knowing the format and structure of questions fed into the system for it to be recognized as tokens. It likewise follows a strict set of rules for querying, even though in natural language.

3.1 Model of the System

Our querying model includes four main modules: Database, Intelligent/Flexible Query Components, the Flexible Query Language Processor, and the Login/Dialog Component. Figure 1 depicts the conceptual model of the system and inter-relations of its main components.

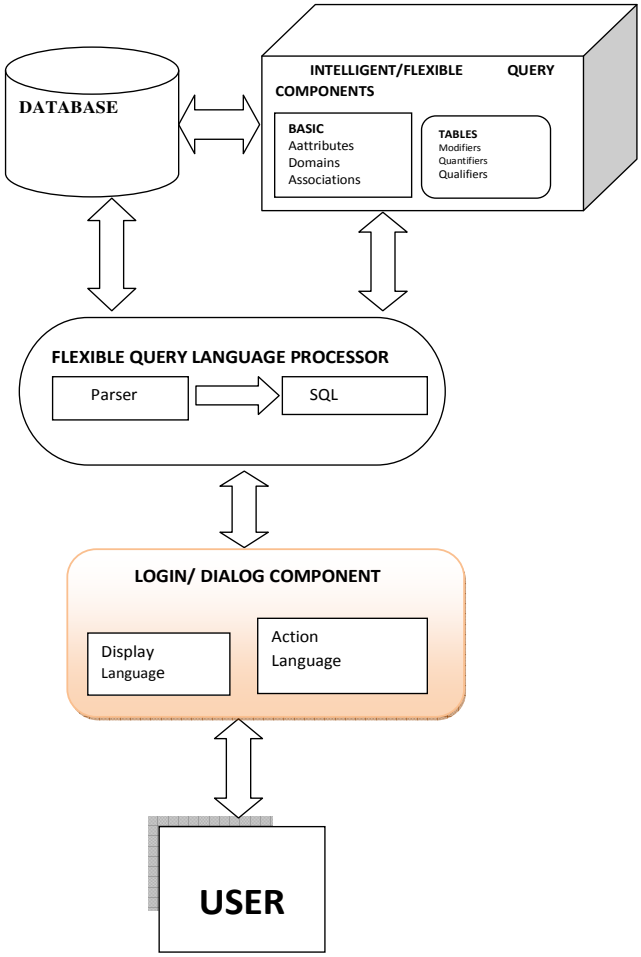

Figure 1: Model of the Proposed System

As the **user** makes use of the system, he is first greeted by the **Login/Dialog component** which is responsible for the authentication of the user who is expected to have provided the correct credentials, and then the **Flex ible Query Language Processor** then takes over after the authentication and connects to the remote servers in the network as well as clients hosting the database and retrieves the data structures and information on the host system, then passes on to the Intelligent/Flexible Query Component which contains two major subcomponents namely; **Basic** and **Tables**. The Basic along with its attributes, Domains, and Associations is responsible for connecting to Server as well as picking out all the databases on the server for selection as well as the tables in such databases, and then the Table subcomponent then selects the rows and columns on the tables presented by the Basic and presents to the user for querying.

In our proposed system architecture, we identify four main components of the system and their interactions with the Local Transaction Management System (LTMS) and Local Database Management System respectively. Our User-Friendly Query Interface operates at the application layer of the OSI model, the software is introduced to provide the interface with remote sites.

The User Interface manager module is responsible for the translation of queries into global form if necessary, determines the location of the data referenced in the queries, and passes control to the Local Transaction Management system (LTMS) if the transaction is local only or to the Global Transaction Analyzer if the transaction needs access to remotely located data as determined by the JDBC API. This Module is also responsible for gathering all user results generated during transaction execution and presenting the results to the user and this actions are also executed by the JDBC API. Transaction Plan Generator (TPG) is the module responsible for generating an execution graph for optimizing the performance of the arriving transactions. The Global Transaction Execution Monitor is the module responsible for receiving the plan generated by the TPG and responsible for the initiation, execution, and integrity control (synchronization, reliability) of the transaction plan.

www.ajocict.net

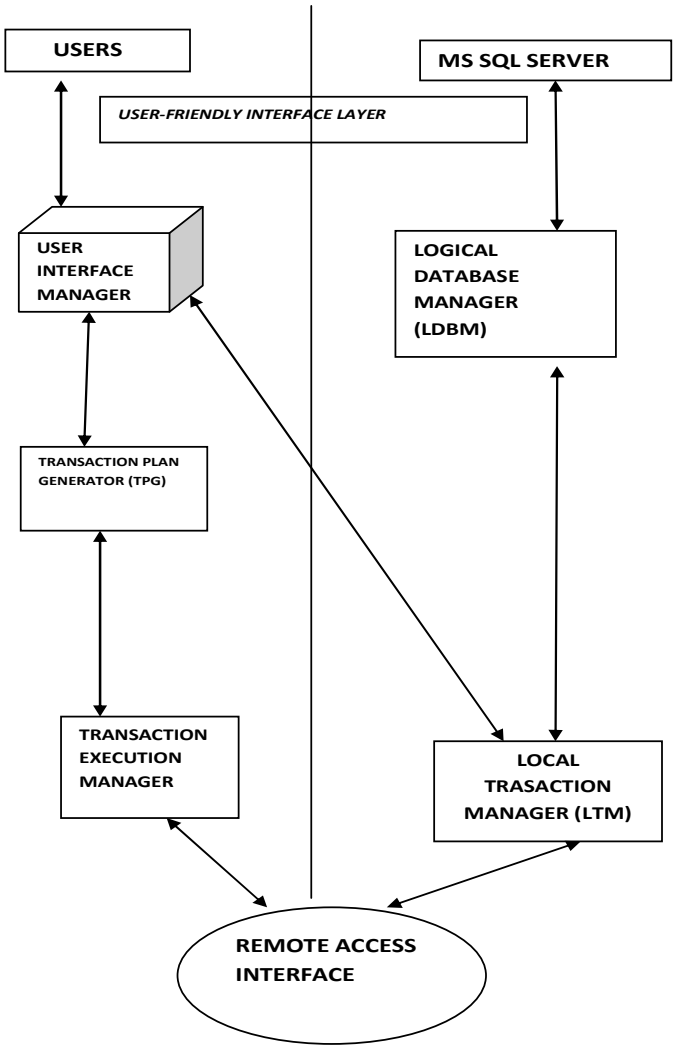

Figure 2: Architecture of the System

4. IMPLEMENTATION

To verify the efficiency of the proposed system, we conducted a simulation test of systems running MS server 2005, created sample databases, tables and data on each of the host, and deployed the proposed system for the network. For the wide area network, our university's intranet was used. Laptops were used as nodes with each laptop signifying a host server. Because of the financial implications, we opted to telnet instead of more secured protocols like the secured shell (SSH). We were able to interface each of the databases on the host network servers, select available databases, and automatically select the tables present in the database, and could query the columns of data present in the tables one by one, and all at once as deemed fit all without writing manual queries, but just at the click of the mouse.

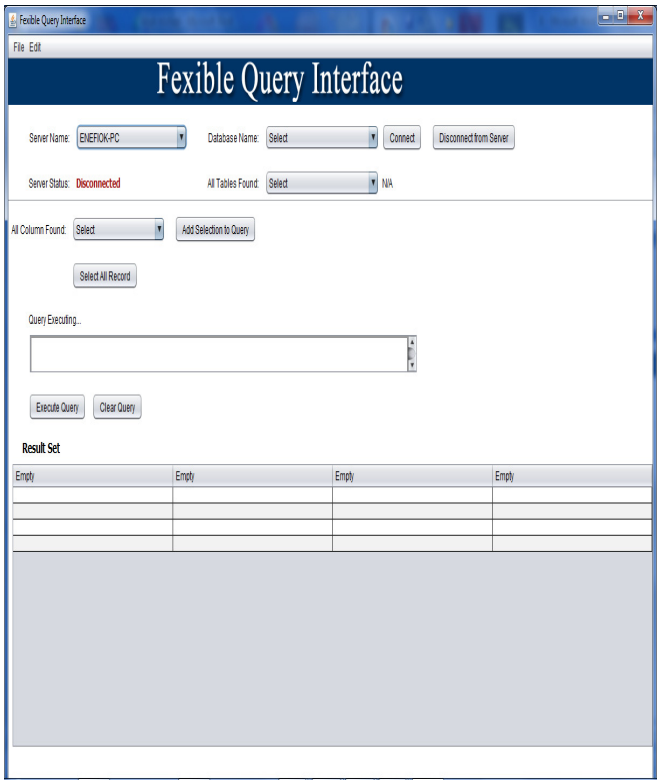

Figure 4. Flexible Query Interface module

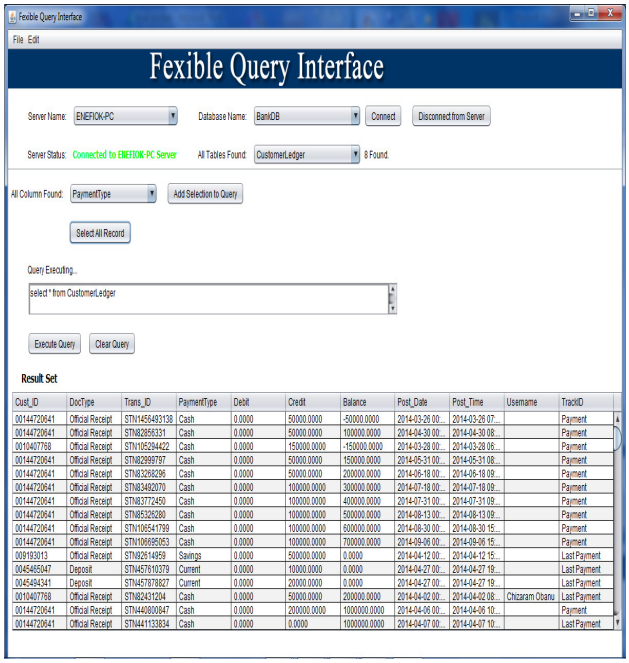

Figure 5 The Flexible Query Interface with the Query Result

4.1 The flexible Query Interface Module.

Figure 4 shows Flexible Query Interface module. The FQI provides the user with the server name of all the servers in the system. From the list of server provided by the system the user can now select the required server. The sever status will indicate connected immediately the user is connected to the server. All the created databases in the server will appear for the user to select the required database. From the FQI, there is a button that list out the entire tables that are found in database. The user is required to select from the list of tables and FQI still gives the user an alternative to create tables when the desired table is not available. The user then executes query if the desired query is already written and there is also an alternative for the user to write his/her own query.

4.2 The Flexible Query Interface with the Query Result

Figure 5 shows the flexible query interface (FQI) with the query result. After successfully executing a query for queries that are often executed, you can save such query as a script file with the save script button for further execution. The execute query from script button allows you to execute queries that are saved on script file. After every successful execution of queries (e.g THE SELECT STATEMENTS) the result are displayed on the Result Set Table but for other queries like the DDL or DML, a message dialog box displays the success/error message.

4.3 Experimentations and Results

Table 1: The Result Set for the Query (Select*from accounts)

Account_No	Cust_ID	Account_Type	Branch	Balance
0042321962	CST1	Corporate		
0061280944	CST2			
0061310530	CST3			
0061341126	CST4		Head-Office	
0061386846	CST5		Head-Office	
0061460477	CST6		Head-Office	
0061484299	CST7	Savings	Head-Office	5000
00138299430	CST8	Savings	N/A	20000
00144720641	CST9	Savings	N/A	50000
0010407768	CST2	Current	Head-Office	50000
009193013	009193013	Savings	N/A	500000
0045465047	0045465047	Current	N/A	10000
0045494341	0045494341	Current	N/A	20000

Table 2 the Result Set for the query (select*from customerLedger)

Cust_ID	DocType	Trans_ID	PaymentT...	Debit	Credit	Balance	Post_Date	Post_Time	Username	TrackID
00144720...	Official Re...	STN1456...	Cash	0.0000	50000.00...	-50000.00...	2014-03-...	2014-03-...		Payment
00144720...	Official Re...	STN8285...	Cash	0.0000	50000.00...	100000.0...	2014-04-...	2014-04-...		Payment
00104077...	Official Re...	STN1052...	Cash	0.0000	150000.0...	-150000.0...	2014-03-...	2014-03-...		Payment
00144720...	Official Re...	STN8299...	Cash	0.0000	50000.00...	150000.0...	2014-05-...	2014-05-...		Payment
00144720...	Official Re...	STN8326...	Cash	0.0000	50000.00...	200000.0...	2014-06-...	2014-06-...		Payment
00144720...	Official Re...	STN8349...	Cash	0.0000	100000.0...	300000.0...	2014-07-...	2014-07-...		Payment
00144720...	Official Re...	STN8377...	Cash	0.0000	100000.0...	400000.0...	2014-07-...	2014-07-...		Payment
00144720...	Official Re...	STN8532...	Cash	0.0000	100000.0...	500000.0...	2014-08-...	2014-08-...		Payment
00144720...	Official Re...	STN1065...	Cash	0.0000	100000.0...	600000.0...	2014-08-...	2014-08-...		Payment
00144720...	Official Re...	STN1066...	Cash	0.0000	100000.0...	700000.0...	2014-09-...	2014-09-...		Payment
009193013	Official Re...	STN8261...	Savings	0.0000	500000.0...	0.0000	2014-04-...	2014-04-...		LastPay...
00454650...	Deposit	STN4576...	Current	0.0000	10000.00...	0.0000	2014-04-...	2014-04-...		LastPay...
00454943...	Deposit	STN4578...	Current	0.0000	20000.00...	0.0000	2014-04-...	2014-04-...		LastPay...
00104077...	Official Re...	STN8243...	Cash	0.0000	50000.00...	200000.0...	2014-04-...	2014-04-...	Chizaram ...	LastPay...

5. DISCUSSION OF RESULTS

In table 1 the Result Set for the query (select*from accounts) was displayed by just clicking a button. The columns that were displayed include the account number, customer identity, account type, branch and balance. In table 2 the Result Set for the query (select*from ComputeProfit) was displayed by just clicking a button. The columns that were displayed include the account number, account type, profit and DateUpdated.

6. CONCLUSION

A user friendly interface for a distributed database for an efficient and flexible database query processing model has been developed. The model represents the first step towards the support of more diverse and richer set of queries and presents the techniques for flexible query processing. We also described the algorithms of query processing unstructured system to obtain high quality answers while minimizing the communication cost.

REFERENCES

1. Akbarinia R., Pacitti, E., and Valduriez P.(2007), "Best Position Algorithms for Top-K Queries" Proceeding of the 33rd International conference on Very Large Database '07, Vienna. Pg 142-151
2. Ben K., Kazar O., and Caplat G., (2011) "intelligent query processing for semantic interoperable information systems". 16th International Conference on Computer Modelling and Simulation. 978- 982
3. Benharzallah S, Kazar O, and Caplat G. (2013) "Intelligent Query Processing for Semantic Interoperable Information Systems" The 5th International Conference on Information Technology. Vol. 14(1): 67–78.
4. Davis U.C, (2014). "Optimizing Query Processing in Catch Aware Wireless Sensor Network". Information Systems vol 36(2) 267-291.
5. Gharib M., Mohamed R., and Zahraa E., (2013) "Intelligent Multidimensional Database Interface" International Journal of Scientific & Engineering Research, Volume 4, 202-212
6. Hallett C., (2006) "Generic Querying of Relational Databases using Natural Language Generation Techniques", Proceedings of the Fourth International Natural Language Generation Conference, pages 95-102.
7. Ilyas I, Beskales G., Mohamed S., and David R. (2008) "A Survey of Top-kQuery Processing Techniques in Relational Database Systems" ACM Computing Surveys, Vol. 40, 58-67
8. Kacprzyk, J., and Zadrozny, S. (2001). "Computing with words in intelligent database querying: standalone and Internet-based applications, Information Sciences" 134, Elsevier, pp.71-109
9. Ndueso, Etukudo Ekefre, and Asagba Oghenekaro, (2014), "A Database Query Processing Model in Peer-to-Peer Network" Journal of Applied Science and Environmental Management. Vol 8, 249-253.
10. Neelu N., Sanjay S., and Mahesh M. (2009) "Design of an intelligent layer for flexible querying in database" Internat**i**onal Journal on Computer Science and Engineering Vol.1(2), 30-39.
11. Neelu N., Sanjay S., and Mahesh M.(2010), "An Intelligent Interface for relational databases" International Journal on Computer Science and Engineering Vol.1(5), 330-340
12. Nittaya K. and Kittisak K., (2012) "Semantic-based query answering supported association patterns and materialized views" International Journal of Database Theory and Application. Vol. 5, 62-71.
13. Oussama Tuli, Minyar S., and Habib O. (2001) "Intelligent Database Flexible Querying by Approximate Query Processing (AQP)". Transactions on Large-Scale Data- and Knowledge-Centered Systems. Pages 1-27.
14. Paolo Dongilli and Enrico Franconi (2006). "An Intelligent Query Interface with Natural Language Support" American Association for Artificial Intelligence